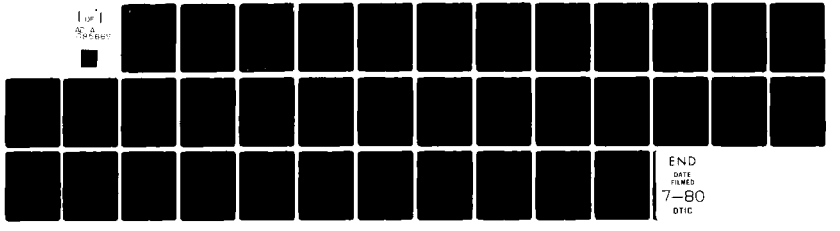
AD-A085 665 VIRGINIA POLYTECHNIC INST AND STATE UNIV WASHINGTON --ETC F/G 9/2
CMS SOFTWARE NOTEBOOK, FIRST EDITION,(U)
JUL 79 R J ORGASS AFOSR-79-0021
UNCLASSIFIED VPI/SU-TM-79-6 AFOSR-TR-80-0448 NL



END
DATE FILMED
7-80
DTIC

EXTENSION DIVISION

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

CMS SOFTWARE NOTEBOOK*†

(First Edition)

edited by

Richard J. Orgass

Technical Memorandum No. 79-6

July 31, 1979

ABSTRACT

A brief description of the software that is available in the Computer Science Library on CMS. Additional contributions are solicited and directions for contributing software are given.

---

* Work supported in part by the Air Force Office of Scientific Research, Air Force Systems Command, under Grant No. AFOSR-79-0021.

† The information in this document is subject to change without notice. The editor, the authors, Virginia Polytechnic Institute and State University, the Commonwealth of Virginia and the United States Government assume no responsibility for errors that may be present in this document or in the programs described here.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: AFOSR-TR-80-0448	2. GOVT ACCESSION NO.: AD-A085 665
	3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): CMS SOFTWARE NOTEBOOK, First Edition	5. TYPE OF REPORT & PERIOD COVERED: Interim
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): Richard J. Orgass	8. CONTRACT OR GRANT NUMBER(s): AFOSR-79-0021
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Virginia Polytechnic Inst. & State University, Department of Computer Science, Washington, DC 20041	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 61102F 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Office of Scientific Research/NM, Bolling AFB, Washington, DC 20332	12. REPORT DATE: July 1979
	13. NUMBER OF PAGES: 36
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

VPI/S2-TM-79-6

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A brief description of the software that is available in the Computer Science Library on CMS. Additional contributions are solicited and directions for contributing software are given.

411731

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

80 6 11 021

PREFACE

This manual together with the CMS userid CSDULLES is designed to provide an orderly and relatively painless way of making software generated within the Department available to potential users.

I have the feeling that many of us have programs that would be useful to others in the Department if we only knew that they existed. Some of us have written programs that may eventually be included in the public libraries of the computing center but which are in some state of being installed -- a long and slow process. This Departmental library is an attempt to provide for wider use of software while it is being installed by the computing center.

I have undertaken the job of maintaining this library at least until someone else volunteers to take the job -- I'll be happy to quit immediately. In order to make the maintenance job tolerable, I've written down some standards for material to be included in the library.

This first edition of Software Notebook serves to introduce the concept of a Departmental library. I have included documentation of some of my programs that may be useful both to bring this programs to your attention and to provide examples of library submissions.

If there is no visible interest in the library by the end of this calandar year, I will terminate the Software Notebook as being a waste of effort.

The first two chapters of the notebook describe the library conventions and the remaining chapters describe software in the library. There are four more chapters for complete programs, utility programs, procedures or classes and execs.

ACCESSING THE LIBRARY

All of the files needed to use programs in the library are stored on the 191 disk of userid CSDULLES. All of the directions given in the notebook assume that this disk is a read only extension of the user's A disk.

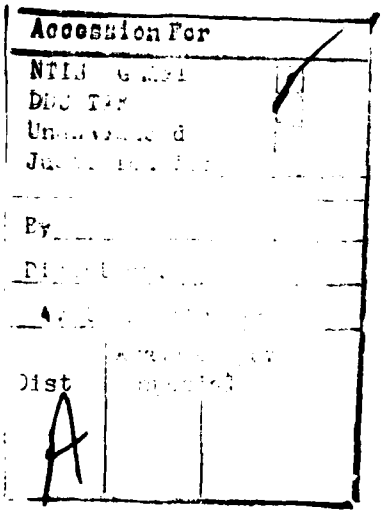
This state of affairs can be arranged by executing the following commands:

```
cp link csdulles 191 250 read all
access 250 d/a
```

After these commands are executed, you can learn about the command:

```
help csdulles
```

and following the directions given when this command is executed.



CHAPTER I

SUBMISSION REQUIREMENTS

The intent of the submission requirements is that there should be an easy way to access programs in the library without complicated directions. I've also added some items to make it easier for me to maintain the library and the Software Notebook. These requirements are summarized below.

Help File

Each program made available via CSDULLES must have a help file. If the name of the program is FOO, this will be a file FOO MEMO. (This file type makes it possible to print the help file on line using the HELP command.)

This help file will contain directions for running the program and a brief description of the processing performed by the program. The help file may refer the reader to a published Technical Report or Technical Memorandum for additional information.

The help file will contain the author's name and address and, hopefully, CMS userid for trouble reports, etc.

The help file serves both as directions for using the program and as an advertisement for the program. Help files will be included in the Software Notebook.

Executable Code

For complete programs, a module for the program will be placed on CSDULLES. If there is some obscure reason for loading the program each time it is executed, text files may be substituted for the module. One obvous reason is: "The program works beautifully if it is loaded just before execution but a module simply won't work." I have encountered such problems in the past and appear to have gotten around old ones but I'm not at all confident that I can always do this.

For a separately compiled procedure or SIMULA class, a TEXT or SIMCLASS file will be placed on CSDULLES.

Execs

If a program is executed via an EXEC, the necessary execs will be placed on CSDULLES. Execs should be designed under the assumption that the 191 disk of CSDULLES is a read only extension of the user's A disk.

Data Files

If a program on CSDULLES requires standard data files each time the program is executed, these data files will be placed on CSDULLES. For example, SIMED, an indentation and reformatting program for SIMULA source files reads a list of identifiers before processing its input text. The file containing these identifiers would also be on CSDULLES.

Source Files

Userid CSDULLES has a 192 disk which will be used to contain source files for programs and procedures that are on the 191 disk. Each submission for the library will be accompained by source file(s) in packed format.

Source files will be maintained for two reasons: This disk area provides a convenient place to save the files while they are not being used. Second, if source files are available, other users are able to modify programs as needed.

The help file is to identify source files and the source language for prospective users.

Additional Documentation

If the help file does not contain a complete description of the program, the help file may refer the reader to other documents. However, the help file must contain directions for obtaining a copy of this additional documentation.

Script File

The Software Notebook will contain help files for all of the executable code placed in CSDULLES. I simply don't have the resources to keyboard all of this text and, therefore, request a SCRIPT file that will generate the help file. This file should not contain any SCRIPT commands that specify the page layout or headings. Please use .cp brackets around text that should remain on the same page.

I don't know how to state this last requirement in the form of specific directions so please try to understand my request. I'd like to have SCRIPT in the same state before and after including your file in the next edition of the Software Notebook. Please don't permanantly move margins, etc. If you use .ad +7, please include a .ad -7 at the end of the file.

Installation

. If you have a piece of software to contribute, please prepare the files in your own userid and then send a message that includes directions for accessing the files to userid ORGASS; please don't disk dump the files.

Before installing a piece of software, I will check the documentation but I will not necessarily test the software itself. I don't want to get into the position of testing software -- life is too short!

Removal

If the amount of disk space used to maintain the library becomes too large, I will conduct a survey to find out which software is being used and delete material that is not of interest.

Before deleting material, I will make a serious attempt to contact the author so that a copy may be preserved. If CMS mail about this is ignored for an extended period of time, I will disk dump the files to the author.

Security

The disks of CSDULLES are read to all and contributing a program to the library implies permission for all users of the system to use the program.

The disks of CSDULLES may not be written by any user and I am planning to keep the login password confidential to limit the risk of deletions of files. While maintaining the library, I will take every precaution to avoid deleting files but, obviously, I cannot offer absolute gurantees. I store my only copy of some software on CSDULLES and am hoping that this will not be a mistake.

CHAPTER II

DESIRED CONVENTIONS

In this chapter I describe some conventions for the behavior of programs that I have found useful. The basic idea is that I can find out what kind of output is written on the terminal and what kind of input is expected. Except for the first requirement, the other conventions are optional.

Gaining Access

A user of this library gains access to the programs in the library by executing the following commands. (I'm assuming that the user will refer to the library disk as disk E; any other letter will do.)

```
cp link csdulles 191 250 read all
access 250 e/a
```

The import of this access mechanism is that the library disk is a read only extension of the A disk so auxiluary execs, etc. can be found using the usual access rules and no special code.

Terminal Prompts

I find it very helpful to have different terminal prompts for different kinds of input. The convention that I use is taken from TOPS-10 and I've become accustomed to it over the years and find it very helpful.

CMS at monitor level prompts with the character period (.). The effect of this is that I know CMS is expecting input when a period appears. This means that a program has successfully completed execution.

A running program prompts for primary input with the character asterisk (*). When this prompt appears, I know that the program is expecting input that is required to continue processing.

A running program prompts for secondary input with the character sharp (#). For example, one of my file managing procedures will permit a user to enter the CMS subset to look for a missing file when it is not possible to open the specified file. When in this subset, the program prompts with the character # to indicate that input is not directed to the main processing.

Execution Messages

I prefer to have each program begin execution with a greeting line that identifies the program and the version number. The date of the version is sometimes helpful too.

When a program has encountered a fatal error in the input data, the error message written to the terminal begins with the character question mark (?). This kind of message is issued when processing is abandoned.

When a program encounters an error in the input data that precludes a correct answer or when it is necessary to warn the user of some condition, this message is preceded by the character percent (%). The presence of this character at the beginning of a message means that something has gone wrong but processing continues. The output may not be correct.

Advisory messages are written inside square brackets. It's often a good idea to provide messages about the progress of a computation. Such messages are enclosed in square brackets. For example, LPTSPL spools a group of files for off-line printing. When each input file has been spooled, a message indicating that the file has been processed is written to the terminal. At the end of orderly execution, I always have programs print a message to this effect. Messages of this type are enclosed in square brackets.

When many pieces of software are used together, it is often difficult to identify the source of a message. I find that it is a good idea to precede the message text with the name of the program or procedure that wrote the message. This convention is used by MULTICS system software.

It isn't very difficult to change the terminal prompt while a program is in execution. For example, in Fortran or RATFOR, the statement

```
call syscal('CP TERM PROMPT *',16,iret)
```

will change the terminal prompt to asterisk (*). [The last parameter contains the return code when CP or CMS commands are executed.] In SIMULA, this is accomplished as follows:

```
EXTERNAL ASSEMBLY PROCEDURE cpcommand;
. . .
command :- copy("TERM PROMPT *");
cpcommand(command)
```

Input and Output Files

If a program requires a single file as input, this file should have a default file type, <ft>. The input file name, <fn>, should be the only parameter to the exec that runs the program. Output files should have <fn> as their file name and different file types. For example, the default file type for the RATFOR processor is RATFOR. The processor output files are <fn> FORTRAN (optionally) and <fn> TEXT. In addition, if there are error messages from RATFOR, these messages are written to the terminal and to file <fn> ERROR.

CRT terminals have a very small screen and it's often useful to be able to refer to error messages after they have fallen off the screen. An ERROR file that also contains these messages is very helpful when correcting programs.

Some programs are able to process more than one file at a time. In this case, the list of file names should be read from the terminal as a list (hopefully separated by commas). If there are a number of different options for the processing to be performed by the program, the user should be prompted with questions. A good example of this kind of dialog is the preliminary dialog when SIMED begins execution.

A long sequence of questions is often a trial when using a program. It's a good idea to provide a way of escaping from a long list of questions by selecting default answers. A good way to do this is to terminate an answer with the character <escape>. If the last character of a response is <escape>, then default answers to all remaining questions are selected without further interaction at the terminal.

An EXEC with a long list of parameters quicky discourages prospective users -- it's too hard to remember the order of the arguments. Can you reliably describe all of the options for the command CP TAG?

CHAPTER III

COMPLETE PROCESSORS

This chapter contains brief descriptions of complete programs that perform specific tasks.

The two programs in the current edition are the RATFOR processor and a gradebook workspace for VSAPL. In later editions, I plan to add an SLR(1) parser generator and an incremental program verifier. These programs are not yet completly operational on CMS.

The current contents of Chapter III are:

RATFOR

A Rational Alternative to Fortran

RATFOR is a fairly popular preprocessor for Fortran. It provides modern control structures as well as a pleasant syntax for Fortran programs. Many of the truly irritating syntactical conventions of Fortran are avoided in RATFOR and the resulting Fortran code is of good quality. Note that RATFOR will not write any non-standard Fortran unless you write it into the program directly.

Any of the published documentation on RATFOR combined with the material in this file is adequate to use the preprocessor. CMS RATFOR differs from other versions in that lower case letters are mapped into upper case letters except in quoted strings and Hollerith constants. Both single and double quotes may open a quoted string; the next occurrence of the same quote ends the string. Detailed documentaton for this CMS implementation may be obtained by contacting:

Richard J. Orgass
Department of Computer Science
VPI&SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid: ORGASS

Please specify if you want the user's manual or the systems manual. The latter is designed for readers who wish to modify the preprocessor and does not contain information that is needed by general users.

Using RATFOR

The RATFOR preprocessor accepts input files with fixed length records and 80 column record length. Tabs are equivalent to blanks so that tabs can be used to provide indented program text that visually extends beyond column 80.

Input files to RATFOR must be of file type RATFOR and located on the user's A disk or a read only extension of this disk. If you have created a file <fn> RATFOR, this file can be converted into a text file for execution by executing the command:

```
ratfor <fn>
```

This will invoke the RATFOR processor and then the Fortran G compiler. If there are no errors, only a file <fn> TEXT will be produced.

If there are RATFOR errors, error messages will be printed on the terminal and written to file <fn> ERROR. The Fortran compiler will not be invoked.

If there are no RATFOR errors but there are Fortran errors, at least two additional files will be produced: <fn> FORTRAN and <fn> LISTING. [There may also be a file <fn> TEXT if one is produced by Fortran.] The listing file will only contain error messages and not the complete text of the program.

Note that the Fortran files produced by RATFOR are very difficult to read; there are no comments and no redundant spaces. These files are only intended for compilation and not for human consumption.

Entering the command:

```
ratfor ?
```

will print this text on your terminal.

Additional Options

Executing the command:

```
ratforh <fn>
```

will invoke the Fortran H compiler instead of the Fortran G compiler to translate the RATFOR program into executable code.

Executing the command:

```
ratno <fn>
```

will invoke the RATFOR preprocessor and will not subsequently invoke any Fortran compiler. If there are no RATFOR detected errors, a single file, <fn> FORTRAN will be created; it contains the Fortran that corresponds to the input RATFOR. If RATFOR detects errors, error messages will be written on the terminal and into file <fn> ERROR.

Both of these commands accept ? as an argument to print this text.

Availability

All of the files needed to use RATFOR are part of the public library of the Computer Science Graduate Program in Northern Virgina. This library is the A disk of userid CSDULLES. To use RATFOR, execute the following commands:

```
cp link csdulles 191 330 read all
access 330 d/a
```

The RATFOR processor assumes that this disk is a read only extension of your A disk; it will not function correctly if this is not the case!

Relevant Files

The following files, located on the 191 disk of CSDULLES are part of the RATFOR processor:

```
RAT       MODULE
RATFOR    EXEC
RATFORH   EXEC
RATNO     EXEC
QUERYFIL  EXEC
RATFOR    MEMO
```

The RATFOR source for the preprocessor is available to individuals who might wish to modify the program for their own use. Please contact userid ORGASS if you wish to secure a copy of the source code.

GRADE

An APL Gradebook Workspace

This VSAPL workspace provides a set of functions for maintaining the data that is usually recorded in an instructor's grade book as well as functions for computing grades and printing reports. It is not necessary to be familiar with APL to use the workspace but an experienced APL programmer can easily extend the workspace to meet individual needs.

Documentation

This workspace has a variety of capabilities that are described in detail with the help of examples in:

R. J. Orgass and M. D. Parker. "A Gradebook Workspace", Technical Memorandum No. APLAD9a, Department of Computer Science, University of Arizona, September 1, 1977.

Copies may be obtained from the author at the address below.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The VSAPL workspace and the help file are stored on the 191 disk of userid CSDULLES (read password ALL). The files are:

GRADE VSAPLWS
GRADE MEMO

The source text is included in the VSAPL workspace.

CHAPTER IV

UTILITY PROGRAMS

The intended contents of this chapter is brief descriptions of utility programs that perform some useful task.

In this edition, Chapter IV contains brief descriptions of the following programs:

SIMED	SIMULA indentation and reformatting program
SPOOL	Print files on hardcopy terminals
LPTSPL	Print files on lineprinter
TPRINT	Simulate line printer on hard copy terminal
TRANS	Repair ASCII to EBCDIC translation performed by IBEDIT when reading ASCII tapes

SIMED -- IBM Version 1.0

SIMULA EDITOR AND INDENTATION PROGRAM

The SIMED program converts SIMULA source program files to change the appearance and representation of the program. The program is able to indent the program (to improve readability) and convert reserved words and standard and user identifiers to:

(1) UPPER CASE
(2) lower case
(3) Edit Case (First character upper case, remainder lower.)

Lines which become too long at indentation are cut at an appropriate position. In the case where no proper cut can be done (long text constants, for example), a warning message is issued.

The program is executed by entering the CMS command:

simed

Inputs describing the processing to be performed are requested by terminal prompts at run time. If there is a default answer to a question, this response can be selected by simply entering carriage return. Default answers are printed as part of the question enclosed in slashes (/).

For a further explanation of the question, respond with the character question mark (?). After the explanation is printed, you will be asked the question again.

The following information is requested from the user:

(1) The file name for the source program file. The file type must be SIMULA. If the file name is followed by the character <escape>, ASCII 27, the default answers to questions (2) to (11) are selected without further questions.

(2) The file specification for the output file. The default answer is <fn> SIMED. If the last character of this response is <escape>, then the default answer to questions (3) to (11) is selected without further questions.

(3) The maximum record length at output. The default response is 72. This provides for 80 column records with columns 73 to 80 blank and is compatible with proposed extensions of the IBM SIMULA compiler.

(4) The number of positions of indentation to be provided for each block or compound statement encountered in the program. If the response is 0, no indentation is performed. If the response is a negative number, the absolute value of the number is used for indentation and leading blanks and tabs are retained in the output file. The default answer is 0.

(5) The rightmost position on a line where indented text may begin. If an indented line would begin beyond this position, the starting position of the line will be taken modulo the entered value. The default response is 52.

(6) The next query asks if tabs may be used in the indentation. If the answer is yes, strings of blanks at the beginning of lines are replaced by the appropriate number of tabs and spaces. It is assumed that tabs occur every eight spaces in accord with the ANSI and ISO standard. Since the SIMULA compiler and most CMS software does not support tabs, the default answer is no.

At this point the program will print a list of the conversion modes on the terminal together with a mode number. These mode numbers are to be used to answer questions (7) to (11).

(7) Enter the conversion mode for reserved words. All of the reserved words in the program text (e.g., BEGIN, IF, WHILE, ...) will be printed in this mode. The default mode is 1 (upper case).

(8) Enter the conversion mode for standard identfiers. All of the SIMULA standard identifiers in the program text (e.g., Infile, Detach, Outfile, will be printed in this mode. The default mode is 3 (edit case).

(9) Enter the conversion mode for user identifiers. All of the user declared identifiers in the program (as well as undeclared identifiers) will be printed in this mode. The default mode is 2 (lower case).

(10) Enter the conversion mode for comments and option statements. The text of all comments (but not the keyword COMMENT) will be printed just as they appear in the input file.

(11) Enter the conversion mode for text constants. All text constants in the program will be written in this mode. The default answer is mode 0 (no change). Other modes are useful primarily when converting SIMULA programs for execution with UNIVAC and CDC SIMULA.

Warning

Cut lines will not be indented properly if they contain BEGIN or END. In addition, cut text constants will not be properly indented.

SIMED treats the national letters of the ISO standard as letters. This means that the characters @, $, ~, `, [,], { and } are letters.

Authors

SIMED was written for DEC-10 SIMULA by Mats Ohlin of the Swedish Research Institute of National Defence. It was adopted for use with IBM SIMULA by:

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The source, help and data files for this program are stored on the 191 disk of userid CSDULLES (read password ALL). The files are:

```
SIMED    MODULE
SIMED    DATA
SIMED    HELP
```

The program is executed directly without an EXEC.

The source and documentation files for this program are stored on the 192 disk of userid CSDULLES (read password ALL). The files are:

```
SIMED    SIMULA
SIMED    SIMED
SIMED    SCRIPT
```

Other Applications

SIMED reads a list of keywords and standard identifiers from file SIMED DATA. SIMED can be modified to perform the same function for programs written in other block structured languages by simply changing the list of identifiers to reflect the syntax of other languages. Easy changes are for PASCAL and SAIL. PL/I can probably be indented using the program but I don't know if this has been tried.

SPOOL -- Version 1.1

(Program to Spool Source Files to Hardcopy Terminals

This program prints CMS files with record lengths less than or equal to 122 characters on ASCII hard copy terminals. The program was designed as a substitute for a line printer with upper and lower case letters. A companion program, LPTSPL will produce the listings with the same format and pagination using a line printer.

The output of SPOOL is a multi-page listing with a title line at the top of each page that includes the file name and a page number. In addition, by means of control lines, it is possible to control the pagination of listings and to insert titles into the heading lines.

Two listing control commands are recognized by SPOOL: When a line that begins with %PAGE or %page is read, this line is omitted from the listing and the next input line is printed as the first line of the next page of output. When a line that begins with %TITLE or %title is read, the text in columns 10 to 70 of this line are inserted into the heading and a new page of output is started. The %title line is not printed in the output listing.

When it is possible to select a default answer to a question asked by SPOOL, this default answer is printed as part of the question; the default answer is surrounded by slashes (/). To select this answer, simply enter a carriage return.

Execution of SPOOL is begun by entering the CMS command

```
spool
```

Once SPOOL is in execution, it is possible to print an arbitrary number of files on the terminal. File specifications are entered in response to terminal questions, see below.

The first question asked by SPOOL is the depth of the forms in the terminal. The response should be the number of lines that can be printed on a single page of paper. This includes every line of the page! For forms that are 8.5 inches deep installed in a terminal that prints six lines per inch, the appropriate response is 51 lines (the default answer). For forms that are 11 inches deep, the appropriate response is 66 lines for a terminal that prints six lines per inch.

The next prompt asks for the starting page number. When the listing of the first file in the list of file names is printed,

pages before this page number are not printed. Pages beginning with the given page number are printed in the same way that the would appear if the whole file were printed. This capability is provided to make it possible to continue printing a listing after an unexpected disconnection on the telephone line.

The third prompt asks for a list of file names. The response should be one or more file names separated by commas. All of the files whose names appear in this list must be of the same file type and there must not be any spaces in the list of names. The default answer to this question is QUIT and selecting this answer terminates execution of SPOOL.

The fourth prompt asks for the file type of the files whose names were given in response to the second question. The default answer is file type SIMED (because the program was originally written to spool these files).

After these questions, the program asks the user to align the paper in the terminal and then enter return. At this point, the paper should be positioned so that typing will begin on the very first line of the page after a carriage return is entered. After the paper is aligned, enter return and the files will be printed.

After the files you specified in response to the third and fourth prompts have been printed, the third prompt is repeated and you may specify another list of file names. Selecting the default answer to this prompt by entering carriage return will terminate execution. If another list of file names is entered, the fourth prompt (requesting the file type) will be repeated. After an appropriate response, the program will again ask you to align the paper and enter return.

Appropriate precautions are taken to insure that dialog with the program does not appear on the same page with listings.

Restrictions

The current version of this program will not correctly paginate files that contain the character line feed (ASCII 10, EBCDIC 37).

If SPOOL is used with a terminal that does not have hardware tabs, files containing tabs (ASCII 9, EBCDIC 5) will not be printed correctly by the current version of SPOOL. Tabs are not expanded to the appropriate number of spaces.

Warning

It takes a very long time to print files on a 300 baud terminal. If you are printing a large volume of output, it is preferable to use a high speed printer. The program LPTSPL will provide similar printed output using the TN train on a Computing Center printer.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The module and help file for SPOOL are located on the 191 disk of userid CSDULLES (read password ALL). These files are:

```
SPOOL    MODULE
SPOOL    MEMO
```

The program is executed without an EXEC.

The source and documentation files for SPOOL are located on the 192 disk of userid CSDULLES (read password ALL). These files are:

```
SPOOL    SIMULA
SPOOL    SIMED
SPOOL    SCRIPT
```

LPTSPL -- Version 1.0

Program to Spool Files to Lineprinter

This program prints CMS files using a high speed line printer. The printer may be a Computing Center printer with the TN print train (upper and lower case, almost ASCII character set) or any remote station. If files containing upper and lower case letters as well as many of the graphics are printed at a remote station they may look quite peculiar because the remote station will perform some rather strange character translations. The format and pagination of listings produced by LPTSPL are the same as listings produced by SPOOL.

The output of LPTSPL is a multi-page listing with a title line at the top of the page that includes the file name and a page number. In addition, by means of control lines, it is possible to paginate an input file and insert titles into heading lines.

Two listing control lines are recognized by LPTSPL: When a line that begins %PAGE or %page is read, this line is deleted from the listing and the next input line is printed as the first line of the next output page. When a line that begins with %TITLE or %title is read, the text in columns 10 to 70 of this line are inserted into the heading and a new page of output is started. The text will appear in the heading of all subsequent pages. The %title line is not printed in the output listing.

When it is possible to select a default answer to a question asked by LPTSPL, the default answer is printed as part of the question and enclosed in slashes (/). Default answers are selected by simply replying with a carriage return.

Execution of this program is begun by entering the CMS command

```
lptspl
```

When the program begins execution, the user will be asked a sequence of questions concerning the listing to be printed.

The first question asks for the page depth of the output listing in lines. For 8.5 inch deep computer forms printing six lines per inch the appropriate answer is 51; for the same forms at eight lines per inch, the appropriate response is 68. If the desired output is to be paginated in the same way that SPOOL output is paginated, the response to this question should be the same for both programs.

The second prompt asks if the listing is to be printed with the TN train at the Computing Center. If the answer is y or yes, this is the printer that will be used. If the answer is n or no, the user will be asked to provide a remote station number; the output will be printed at this remote station.

The third prompt asks for the number of copies of the listing that are to be printed; the default answer is one copy.

The fourth prompt asks for a list of input file names. The response should be a list of file names separated by commas with no imbedded spaces in the input string. All of the files that appear in this list must be of the same file type. It is possible to print files of different file types in one execution of the program (see below).

The fifth prompt asks for the file type of the files whose names were entered in response to the fourth prompt.

After these inputs, the files specified are spooled to the printer. At the end of each file, an advisory message is printed on the terminal so that there will be an immediately available record of files printed.

Once the specified files have been spooled to the printer, the fourth prompt is repeated. Responding to this prompt with a carriage return will terminate spooling. Otherwise, additional file names may be specified. Following the specification of additional file names, the file type is again requested.

Spooling will continue as described above until the response to the prompt "Input File Name(s):" is carriage return. After this response, the program will ask if the files are to be released for printing. If the answer to this question is y or yes, the files will be printed. If the answer is n or no, the virtual printer is purged so that no files are printed or remain in the virtual printer.

Warnings

If you decide to abort printing of a listing by answering no to the question "Release files for printing", your virtual printer will be purged of all files.

Files containing the character line feed (ASCII 10, EBCDIC 37) will not be printed correctly.

Files containing tab characters (ASCII 9, EBCDIC 5) will not be printed correctly by the current version of LPTSPL.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The execution module and help file are on the 191 disk of userid CSDULLES. The files are:

```
LPTSPL   MODULE
LPTSPL   MEMO
```

The source and documentation files are on the 192 disk of userid CSDULLES. The files are:

```
LPTSPL   SIMULA
LPTSPL   SIMED
LPTSPL   SCRIPT
```

The program is executed directly without an EXEC.

TPRINT -- Version 1.0

Simulate a Line Printer on an ASCII Terminal

This program simulates a line printer on ASCII hard copy terminals with a line length of 132 or greater. Carriage control characters 1 (eject page) and 0 (skip a line) are recognized and processed correctly. Carriage control character + (overprint) is not supported.

The program prompts for inputs and provides a default value enclosed in slashes(/). To select the default input, simply answer with a carriage return.

This program is executed by entering the CMS command

```
tprint
```

The first input is the physical depth of the paper mounted in the terminal in lines. For forms that are 8.5 inches deep in a terminal that prints six lines per inch, this response is 51. For forms that are 11 inches deep in a terminal that prints six lines per inch, this response is 66.

The second query asks for the names of the files to be spooled to the terminal. A list of file names, separated by commas, is the appropriate response. This response may not contain imbedded blanks. Answering with a carriage return terminates execution.

All of the files whose names were entered in response to the second prompt must have the same file type. The third question asks the file type of the files. The default response is LISTING.

The program assumes that files of type LISTING have LRECL = 133. If the file type that you entered is not LISTING, the program will prompt for a record length.

After this sequence of questions is answered, the program will print a message asking you to align the paper and then enter carriage return. Position the paper at the top of a new page and then respond with carriage return. This will start spooling.

After the first group of files have been printed, the program will again ask for input file names. Respond with a carriage return if you are finished. Otherwise, respond with another list of file names as described above.

Warnings

It takes a very long time to print large files on 300 baud terminals. If at all possible, use a line printer.

Files containg tabs and line feeds will not print correctly.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The execution module and help file are stored on the 191 disk of userid CSDULLES (read password ALL). The files are:

```
TPRINT   MODULE
TPRINT   MEMO
```

The source and documentation files are stored on the 192 disk of userid CSDULLES (read password ALL). The files are:

```
TPRINT   SIMULA
TPRINT   SIMED
TPRINT   SCRIPT
```

The module is executed directly without an EXEC.

TRANS -- Version 1.0

Program to Complete ASCII Tape to EBCDIC Translation

When IBEDIT is used to read ASCII tape files, the translation from ASCII to EBCDIC is not completly correct. This program reads CMS files that were read from ASCII tapes and corrects the translation errors.

The translation errors that are dealt with are as follows:

(1) There are two EBCDIC codes for left curly bracket ({) and two EBCDIC codes for right curly bracket (}). The two are indistinguishable on an ASCII terminal but only one of them is printed by the TN train. The translation in IBEDIT selects the non-printable version. This program converts the non-printable version into the printable version.

(2) There are three EBCDIC codes for not (~). The three are indistinguishable on an ASCII terminal but only one of them is printed by the TN train. The translation in IBEDIT selects one of the non-printable versions. This program converts the not-printable versions into the printable version.

(3) There are three EBCDIC codes for vertical line (|). The three are indistinguishable on an ASCII terminal but only one of them is printed by the TN train. The translation in IBEDIT selects one of the non-printable versions. This program converts the non-printable versions into the printable version.

The translation performed by this program is needed for all files containing the above characters if they are to be correctly printed with the TN train. The translation is not necessary if the file is only to be compiled using SIMULA, RATFOR or Fortran and possibly other processors. However, without translation of text and character constants, printed output will be incorrect.

This program is a self contained module and is executed by typing the command:

```
trans
```

The program will prompt for an input file name and input file type. These two items specify the file that is to be translated. The output translated file will have the same file name but the file type is secured from an additional terminal prompt.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The module and help files are stored on the 191 disk of userid CSDULLES. The files are:

```
TRANS    MODULE
TRANS    MEMO
```

The module is self contained and is executed without an EXEC.

The source and documentation files are stored on the 192 disk of userid CSDULLES. The files are:

```
TRANS    SIMULA
TRANS    SIMED
TRANS    SCRIPT
```

CHAPTER V

PROCEDURES AND CLASSES

This chapter is to contain descriptions of separately compiled procedures and classes that can be used in other programs.

In later editions, I hope to add file management procedures for RATFOR and procedures for performing translations from EBCDIC to ASCII and conversely.

Contributions are solicited!

In this edition, the chapter contains:

DIALOG SIMULA class for writing interactive programs and managing files at run time.

DIALOG -- Version 1.0

A SIMULA Class for Writing Interactive Programs

DIALOG is a separately compiled SIMULA class that was designed to serve as a prefix for the main block of SIMULA programs. The class contains procedures for writing programs that interact with users at terminals and for accessing files during program execution. In addition, a number of utility procedures that are part of the DEC-10 SIMULA library are included in the class.

Using the procedure get_infile in class DIALOG it is possible to read any CMS file without knowing any of the properties of the file!

Some of the code in DIALOG depends on the EBCDIC character set as well as details of CMS and CMS SIMULA. It is intended as a machine dependant front end for SIMULA programs. If all references to the environment are made through DIALOG, only DIALOG must be changed when programs are moved to another implementation.

A detailed set of specifications for the procdures in DIALOG is given in:

R. J. Orgass. DIALOG: A SIMULA Class for Writing Interactive Programs. Technical Memorandum No. 79-3, Graduate Program in Northern Virginia, Department of Computer Science, Virginia Polytechnic Institute and State University, May 4, 1979.

Copies may be obtained from the author (see below).

Directions

To use the procedures in DIALOG in a program, the program structure should be as follows:

```
BEGIN
    EXTERNAL CLASS dialog;

    dialog BEGIN

        < text of program using DIALOG >

    END of dialog block;
END of program.
```

If this program is contained in a file TEST SIMULA, the program is compiled with the CMS command:

```
simula test (class dialog <other options>
```

These directions assume that the 191 disk of userid CSDULLES is a read only extension of the user's A disk. These directions replace the directions given in TM 79-3.

Author

Richard J. Orgass
Department of Computer Science
VPI & SU
P. O. Box 17186
Washington, D.C. 20041

CMS userid ORGASS

Relevant Files

The simclass and help files for DIALOG are stored on the 191 disk of userid CSDULLES (read password ALL). These files are:

```
DIALOG   SIMCLASS
DIALOG   MEMO
```

The source and documentation files for DIALOG are stored on the 192 disk of userid CSDULLES (read password ALL). These files are:

```
DIALOG    SIMULA
DIALOG    SIMED
DIALOG    SCRIPT
```

CHAPTER VI

USEFUL EXEC FILES

This chapter contains brief descriptions of EXECs that are generally useful. It is not intended for EXECs that are used to run programs in the library; EXECs of this type should be described in the documentation of the program.

The current edition contains descriptions of the following EXECs:

DPRINT	Print files with TN print train at Computing Center
EDM	Enter CMS editor in a useful way
GO	Execute SIMULA programs from TEXT files
HLP	Extend CMS MEMO to user created help files
KJOB	Delete map files, log hold
QF	Determine LRECL and RECFM of files
QUERYFIL	Place LRECL and RECFM of files in console stack
SIM	Compile SIMULA programs with class dialog and then execute the program

dprint <fn> <ft> [<fm>]

Print the file whose specfication is the parameter of the exec on a printer at the Computing Center using the TN print train. After queuing the file for printing, restore the destination for printed output to remote station 18.

edm <fn> <ft> [<fm>]

Enter the CMS editor with a number of useful options enabled: Tabs are set every eight spaces so that hardware tabs can be used for indentation with the tabs mapped into the appropriate number of blanks. This makes it possible to have files correctly processed by system compilers. Text that is entered into the file is in upper and lower case. It is assumed that the delete character is <backspace> so backspaces are not processed.

To prevent the loss of work when telephone lines disconnect, autosave is set to 10.

The terminal prompt character is set to asterisk (*) at entry and back to period (.) at exit.

go <fn>

If there is a file <fn> TEXT on the A disk, this file is loaded with SIMRTS. The EXEC is designed to execute SIMULA programs. Any additional parameters are run time parameters for SIMRTS.

If there is no file <fn> TEXT the results are unpredictable.

kjob

Delete load map files and execute log hold to continue terminal session.

qf <fn> [<ft> [<fm>]]

Print the file name, file type, file mode, RECFM and LRECL of the file on the terminal. If <ft> is omitted, the first file with <fn> on the search list is used.

queryfil <fn> [<ft> [<fm>]]

Place RECFM and LRECL of the file at the top of the console stack. If <ft> is omitted, the first file in the search list with <fn> is used.

This EXEC is used by RATFOR and DIALOG.

sim <fn>

Invokes the SIMULA compiler for file <fn> SIMULA with compiler options that include simclass dialog. Additional parameters are parameters to the SIMULA compiler.

If the compiler completes with return code 0, the program is loaded as with GO and executed. This is a restricted version of EX in TOPS-10.